析构过程（Deinitialization）

1.0 翻译：bruce0505 (https://github.com/bruce0505) 校对：fd5788 (https://github.com/fd5788)

2.0 翻译+校对：chenmingbiao (https://github.com/chenmingbiao)

2.1 校对：shanks (http://codebuild.me)，2015-10-31

2.2 翻译+校对：SketchK (https://github.com/SketchK) 2016-05-14

本页包含内容：

*析构器*只适用于类类型，当一个类的实例被释放之前，析构器会被立即调用。析构器用关键字 `deinit` 来标示，类似于构造器要用 `init` 来标示。

析构过程原理

Swift 会自动释放不再需要的实例以释放资源。如自动引用计数 (./16_Automatic_Reference_Counting.html)章节中所讲述，Swift 通过 自动引用计数（ARC） 处理实例的内存管理。通常当你的实例被释放时不需要手动地去清理。但是，当使用自己的资源时，你可能需要进行一些额外的清理。例如，如果创建了一个自定义的类来打开一个文件，并写入一些数据，你可能需要在类实例被释放之前手动去关闭该文件。

在类的定义中，每个类最多只能有一个析构器，而且析构器不带任何参数，如下所示：

```
deinit {
    // 执行析构过程
}
```

析构器是在实例释放发生前被自动调用。你不能主动调用析构器。子类继承了父类的析构器，并且在子类析构器实现的最后，父类的析构器会被自动调用。即使子类没有提供自己的析构器，父类的析构器也同样会被调用。

因为直到实例的析构器被调用后，实例才会被释放，所以析构器可以访问实例的所有属性，并且可以根据那些属性可以修改它的行为（比如查找一个需要被关闭的文件）。

析构器实践

这是一个析构器实践的例子。这个例子描述了一个简单的游戏，这里定义了两种新类型，分别是 `Bank` 和 `Player` 。 `Bank` 类管理一种虚拟硬币，确保流通的硬币数量永远不可能超过 10,000。在游戏中有且只能有一个 `Bank` 存在，因此 `Bank` 用类来实现，并使用类型属性和类型方法来存储和管理其当前状态。

```
class Bank {
    static var coinsInBank = 10_000
    static func vendCoins(numberOfCoinsRequested: Int) -> Int {
        let numberOfCoinsToVend = min(numberOfCoinsRequested, coinsInBank)
        coinsInBank -= numberOfCoinsToVend
        return numberOfCoinsToVend
    }
    static func receiveCoins(coins: Int) {
        coinsInBank += coins
    }
}
```

`Bank` 使用 `coinsInBank` 属性来跟踪它当前拥有的硬币数量。 `Bank` 还提供了两个方法， `vendCoins(_:)` 和 `receiveCoins(_:)` ，分别用来处理硬币的分发和收集。

`vendCoins(_:)` 方法在 `Bank` 对象分发硬币之前检查是否有足够的硬币。如果硬币不足， `Bank` 对象会返回一个比请求时小的数字（如果 `Bank` 对象中没有硬币了就返回 `0` ）。 `vendCoins` 方法返回一个整型值，表示提供的硬币的实际数量。

`receiveCoins(_:)` 方法只是将 `Bank` 对象接收到的硬币数目加回硬币存储中。

`Player` 类描述了游戏中的一个玩家。每一个玩家在任意时间都有一定数量的硬币存储在他们的钱包中。这通过玩家的 `coinsInPurse` 属性来表示：

```
class Player {
    var coinsInPurse: Int
    init(coins: Int) {
        coinsInPurse = Bank.vendCoins(coins)
    }
    func winCoins(coins: Int) {
        coinsInPurse += Bank.vendCoins(coins)
    }
    deinit {
        Bank.receiveCoins(coinsInPurse)
    }
}
```

每个 Player 实例在初始化的过程中，都从 Bank 对象获取指定数量的硬币。如果没有足够的硬币可用， Player 实例可能会收到比指定数量少的硬币.

Player 类定义了一个 winCoins(_:) 方法，该方法从 Bank 对象获取一定数量的硬币，并把它们添加到玩家的钱包。 Player 类还实现了一个析构器，这个析构器在 Player 实例释放前被调用。在这里，析构器的作用只是将玩家的所有硬币都返还给 Bank 对象：

```
var playerOne: Player? = Player(coins: 100)
print("A new player has joined the game with \(playerOne!.coinsInPurse) coins")
// 打印 "A new player has joined the game with 100 coins"
print("There are now \(Bank.coinsInBank) coins left in the bank")
// 打印 "There are now 9900 coins left in the bank"
```

创建一个 Player 实例的时候，会向 Bank 对象请求 100 个硬币，如果有足够的硬币可用的话。这个 Player 实例存储在一个名为 playerOne 的可选类型的变量中。这里使用了一个可选类型的变量，因为玩家可以随时离开游戏，设置为可选使你可以追踪玩家当前是否在游戏中。

因为 playerOne 是可选的，所以访问其 coinsInPurse 属性来打印钱包中的硬币数量时，使用感叹号（ ！）来解包：

```
playerOne!.winCoins(2_000)
print("PlayerOne won 2000 coins & now has \(playerOne!.coinsInPurse) coins")
// 输出 "PlayerOne won 2000 coins & now has 2100 coins"
print("The bank now only has \(Bank.coinsInBank) coins left")
// 输出 "The bank now only has 7900 coins left"
```

这里，玩家已经赢得了 2,000 枚硬币，所以玩家的钱包中现在有 2,100 枚硬币，而 Bank 对象只剩余 7,900 枚硬币。

```
playerOne = nil
print("PlayerOne has left the game")
// 打印 "PlayerOne has left the game"
print("The bank now has \(Bank.coinsInBank) coins")
// 打印 "The bank now has 10000 coins"
```

玩家现在已经离开了游戏。这通过将可选类型的 playerOne 变量设置为 nil 来表示，意味着“没有 Player 实例”。当这一切发生时， playerOne 变量对 Player 实例的引用被破坏了。没有其它属性或者变量引用 Player 实例，因此该实例会被释放，以便回收内存。在这之前，该实例的析构器被自动调用，玩家的硬币被返还给银行。

上一篇: 构造过程 (/project/swift/chapter2/14_Initialization.html)
下一篇: 自动引用计数 (/project/swift/chapter2/16_Automatic_Reference_Counting.html)

咸鱼 (http://t.qq.com/xiaopeng50016)
翻译辛苦了

(http://t.qq.com/xiaopeng50016)2015年10月28日 回复 顶 转发

范旋凯 (http://t.qq.com/fan_xuankai)
抢沙发，谢谢作者~~

(http://t.qq.com/fan_xuankai)2015年9月22日 回复 顶 转发